Finding Consistent Categorial Grammars of Bounded Value: a Parameterized Approach

Christophe Costa Florêncio[1] and Henning Fernau[2]

[1] Department of Computer Science, K.U. Leuven, Leuven, Belgium
Chris.CostaFlorencio@cs.kuleuven.be
[2] Universität Trier, FB IV—Abteilung Informatik, D-54286 Trier, Germany
fernau@uni-trier.de

Abstract. Kanazawa ([1]) has studied the learnability of several parameterized families of classes of categorial grammars. These classes were shown to be learnable from text, in the technical sense of identifiability in the limit from positive data. They are defined in terms of bounds on certain parameters of the grammars. Intuitively, these bounds correspond to restrictions on linguistic aspects such as the amount of lexical ambiguity of the grammar.
The time complexity of learning these classes has been studied by Costa Florêncio ([2]). It was shown that for most of these classes, selecting a grammar from the class that is consistent with the data is NP-hard. In this paper existing complexity results are sharpened by demonstrating W[2]-hardness. Additional parameters allowing FPT-results are also studied, and it is shown that if these parameters are fixed, these problems become computable in polynomial time. As far as the authors are aware, this is the first such result for learning problems.

1 Introduction

We consider the complexity of *consistency problems* for some family $(\mathcal{L}_k)$ of language classes of the following form: Given a finite language sample D and some integer k, is there some language $L \supseteq D$ contained in $\mathcal{L}_k$?
For many families of language classes, this type of consistency problem is trivial. Consider for example the class $\mathcal{A}_k$ of languages that can be accepted by a finite automaton with at most k states. In this case we can always answer YES to the consistency problem, since an automaton with just one state exists that accepts Σ^*.
However, this trivial type of reply is no longer possible if the universal language (i.e., Σ^* in the case of string languages) is not (automatically) in each of the language classes of interest. Examples are provided by classes of classical categorial grammars, see [1].
The language families $\mathcal{L}_k$ are usually defined via grammar families $\mathcal{G}_k$. As a variant of the mentioned consistency problem, we may be given a finite set of derivation structures and some parameter k and ask if there is a grammar $G \in \mathcal{G}_k$ that produces those structures (and possibly more). Note that this can be also seen as a special case of the first

problem formulation, if we consider *languages of structures* (formally, labeled ordered trees).
We study the computational complexity of consistency problems both from a classical (P vs. NP) perspective, as well as from the perspective of parameterized complexity.
Since categorial grammars are mainly studied within computational and mathematical linguistics, our results may be especially relevant for researchers from these fields.
In order to keep the paper as self-contained as possible, we will try to provide the necessary background of all the relevant fields. Readers familiar with this material can of course skip these sections. Throughout the paper, we use standard notation from Formal Language Theory. If D is a finite language, then $\|D\|$ denotes the sum of all lengths of all words from D.

2 Categorial Grammars

The classes studied in [3,4] which are the focus of the present paper are based on a formalism for (ε-free) context-free languages called *classical categorial grammar* (CCG). In this section the relevant concepts of CCG will be defined. We will adopt the notation used in [1].
In CCG, each *symbol* (or *atom*) in some given alphabet Σ is assigned a finite number of *types*. In the remainder, we assume Σ to be fixed. This is technically convenient, and makes no difference in the context of learning, since only the subset of Σ that actually appears in the data is relevant for the learner. Types are constructed from *primitive types* by the operators $\backslash$ and $/$. We let Pr denote the (countably infinite) set of primitive types. The set of types Tp is defined as the smallest set satisfying:

1. $\mathrm{Pr} \subseteq Tp$,
2. if $A \in Tp$ and $B \in Tp$, then $A\backslash B \in Tp$.
3. if $A \in Tp$ and $B \in Tp$, then $B/A \in Tp$.

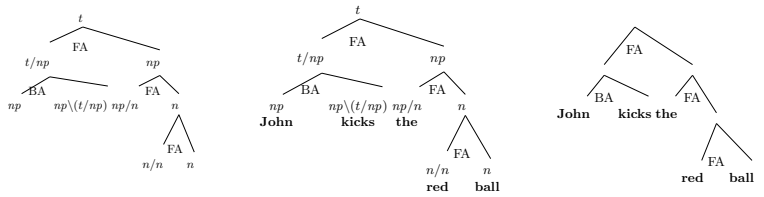
One member t of Pr is called the *distinguished type*, and is considered a constant. In CCG there are only two modes of type combination, *backward application*, $A, A\backslash B \Rightarrow B$, and *forward application*, $B/A, A \Rightarrow B$. In both cases, type A is the *argument*, the complex type is the *functor*. Given an expression of the form A/B ($B\backslash A$), its main operator is '/' ('\').
Grammars consist of type assignments to symbols, i.e., $\texttt{symbol} \mapsto T$, where $\texttt{symbol} \in \Sigma$ and $T \in \mathrm{Tp}$.
A *derivation* of B from $A_1, \ldots, A_n$ is a binary branching, labeled tree that encodes a proof of $A_1, \ldots, A_n \Rightarrow B$. Through the notion of derivation the association between grammar and language is defined. All structures contained in some given structure language correspond to a derivation of type t based solely on the type assignments contained in a given grammar. This is the *structure language* generated by G, denoted $\mathrm{FL}(G)$. The *string language* generated by G, $\mathrm{L}(G)$, consists of the strings corresponding to all the structures in its structure language, where the string corresponding to some derivation consists just of the leaves of that derivation (also known as the *yields*).

The symbol FL is an abbreviation of *functor-argument language*, the derivation language for a CCG that is obtained by suppressing types associated to inner nodes in the derivation (tree). Hence, structures correspond to terms. More precisely, structures are of the form `symbol`, `fa(s1,s2)` or `ba(s1,s2)`, where `symbol` $\in \Sigma$, `fa` stands for forward application, `ba` for backward application and `s1` and `s2` are also structures.

Example 1. The leftmost structure is a derivation for a proof of

$$np, np\backslash(t/np), np/n, n/n, n \Rightarrow t.$$

Note that all tree nodes carry types as labels, and that to inner nodes, in addition, labels BA and FA are associated, which indicate the operations applied at those points in the derivation.
The middle structure shows a CCG parse, which is a derivation where the leaves are labeled not just with types, but also with the lexical items (such as **John**) that these types are assigned to in the grammar used for the parse. The rightmost structure shows the corresponding functor-argument structure.



All learning functions in [1] are based on the function GF. This function receives a sample of structures D as input and yields a set of assignments (i.e., a grammar) called the *general form* as output, which generates exactly D. It is a homomorphism and runs in linear time. GFassigns t to each root node, assigns distinct variables to the argument nodes, and computes types for the functor nodes: if it is the case that `s1` $\mapsto A$, given `ba(s1,s2)` $\Rightarrow B$, then `s2` $\mapsto A\backslash B$. If `s1` $\mapsto A$, given `fa(s2,s1)` $\Rightarrow B$, then `s2` $\mapsto B/A$. When learning from strings, the structure language is not available to the learner, but given a set of strings there exist only finitely many possible sets of structures for the classes under discussion. These are then used to produce hypotheses.
Categorial types can be treated as terms, so natural definitions of substitution and unification apply. A substitution over a grammar is just a substitution over all of the types contained in its assignments. This notion can be used to unify distinct types assigned to the same word. Consider, for example, the following grammar:

$$G = \begin{array}{l} \texttt{a} \mapsto t/A, B\backslash t, C/(E/E), (C/D)/D \\ \texttt{b} \mapsto A, B, t/C, D, (E/E)/D \end{array}$$

(The reader can verify that L(G) consists of just the words `ab`, `ba` and `babb`.) The types t/A and $C/(D/D)$ can be unified to yield the single type $t/(D/D)$, obtained by applying the most general unifier $\sigma\{C = t, A = (D/D)\}$. The type $B\backslash t$ cannot be unified with any of the other types, since they all have '/' as main operator, while $B\backslash t$ has '\' as main operator. The types t/A and $(C/D)/D$ cannot be unified because

their functors are a constant and a complex type, respectively. Finally, $C/(E/E)$ and $(C/D)/D$ cannot be unified because this would fail the *occurs check*: it would require that C is unified with C/D.
We state without proof that $\mathrm{FL}(G) \subseteq \mathrm{FL}(\sigma[G])$ for each substitution σ, see [1] for details.
A CCG G can be hence viewed as a mapping from Σ into finite subsets of types Tp. Accordingly, we can associate a *value function* v that maps $a \in \Sigma$ onto $|G(a)|$, i.e., the number of types that G maps to a. G is *k-valued* [2,1] if $\max_{a\in\Sigma} v(a) \leq k$. The according grammar and language classes are denoted by $\mathcal{G}_{k-\mathrm{valued}}$ and $\mathcal{L}_{k-\mathrm{valued}}$, resp. 1-valued grammars are also known as *rigid grammars*, and denoted as $\mathcal{G}_{\mathrm{rigid}}$. This class is known to be learnable from structures with polynomial update-time, by simply unifying all types assigned to the same symbol in the general form [1]. The other classes originally defined in [3,4] are generalizations thereof.
The class of structure languages that can be generated by grammars from $\mathcal{G}_{k-\mathrm{valued}}$ is written $\mathcal{FL}_{k-\mathrm{valued}}$.

3 Complexity Notions

We assume some familiarity with the basic notions of classical complexity on the side of the reader.
There has been recent interest in the development of parameterized complexity results to allow for a more fine-grained analysis of NP-hard problems. So, a problem (parameterized by k) is in FPT if we can develop an algorithm with running time $O(f(k)p(n))$, where n is the overall input size and k is the (size of the) parameter. f is an arbitrary function (only depending on k but not on n) and p is a polynomial. An algorithm that proves FPT-ness is also called an FPT-*algorithm*, or a *parameterized algorithm*. More details can be found in the monograph [5].
The classical example for a problem in FPT is the VERTEX COVER problem on undirected graphs. So, given a graph G and a parameter k, it is asked if there exists a *vertex cover* C with $|C| \leq k$, where a vertex cover set can be characterized by the fact that, upon removing it together with all incident edges, no edges will remain in the graph.
This approach makes sense in particular if the parameter of interest can be assumed to be small. The hierarchy level k in our formulation of the consistency problem might be such a small parameter: in linguistics, the amount of lexical ambiguity for natural language is assumed to be very small in relation to the number of symbols found in that lexicon. So, we arrive at problems that we call, for instance, $\mathcal{L}_{k-\mathrm{VALUED}}$-CONSISTENCY in order to make the parameter explicit. However, we cannot always hope to find nice parameterized algorithms. More specifically, we can derive as a corollary from [2, Theorem 5.32] (alternatively, [6]):

Corollary 1. *Unless* P = NP*, there is no* FPT *algorithm that decides* $\mathcal{L}_{k-\mathrm{VALUED}}$-CONSISTENCY.

Proof. If this were not the case, there would be an algorithm that decides the consistency problem in time $O(f(k)p(n))$. Setting $k = 1$, we would

arrive at a polynomial-time algorithm that decides if, given a finite set D of strings, there exists a 1-max-valued (rigid) categorial grammar G such that $D \subseteq \mathrm{L}(G)$. This problem is known to be NP-hard. □

This corollary is surely disappointing from a parameterized point of view, since it seems to rule out to use k as a good choice of a parameter for the consistency problem for k-valued categorial grammars.
The proof of [2, Theorem 5.32] makes use of the fact that there is no bound on the size of the alphabet, so we still might hope for better parameterized results when we restrict our attention to languages over alphabets of size three, for example.
As we will see, this hope will not be fulfilled. However, in order to formulate and to establish the indicated result, we need some more notions from parameterized complexity.
As with classical complexity, we need an appropriate notion of reduction to prove hardness results, and some knowledge about classes of parameterized problems that are believed not to possess FPT-algorithms. Actually, there is a whole hierarchy of parameterized problems that is believed to be strict, the so-called W-hierarchy. Usually, its lowest level, W[0], is called FPT (which we have already defined).
A typical W[1]-complete problem is the following one: Given a graph G and a parameter k, is there an independent vertex set I of size k in G? Recall that C is a minimum vertex cover in a graph $G = (V, E)$ iff the subgraph induced by $V \setminus C$ is an *independent set*, see [7].
A typical W[2]-complete problem is the VERTEX COVER problem for hypergraphs, also known as HITTING SET: given a hypergraph G and a parameter k, we ask for a vertex set C of size at most k such that each edge e of G is hit, i.e., $e \cap C \neq \emptyset$ (note that a hyperedge e is simply the set of vertices it connects).
We still need a satisfying notion of reduction in order to define hardness (and completeness) for parameterized complexity classes. Given two parameterized problems P and P' with parameterizations k and k', resp., a *parameterized (many-one) reduction* translates an instance (I, k) of P in polynomial time into an instance (I', k') such that $k' = f(k)$ for some function f. Obviously, if P' is in $W[i]$, $i = 0, 1, 2$, then so is P. So, if P is W[2]-hard and we can provide such a reduction that translates P into P', then P' is W[2]-hard, as well. As an example, consider the reductions presented in Sections 5.3-5.5 from [2] that show NP-hardness of several variants of consistency problems (as we would call those here). These reductions use VERTEX COVER, and the reductions are actually parameterized reductions in the following sense: they show how an instance (G, k) of VERTEX COVER can be transformed in polynomial time into an instance (F, k) of $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY. This is why we originally hoped for FPT-ness results for this type of problems.

4 Results

In [2], only hardness results were shown. We first complement these results by demonstrating membership in NP. This was originally neglected,

since the consistency problem was studied as an aspect of learning problems, specifically of identification in the limit. In this paradigm, the length of the input sequence before convergence is inherently unbounded. Thus, it makes little sense to consider questions such as membership in NP, which would require a polynomial number of steps before convergence.
Lemmas 6.1 and 6.2 from [1] (attributed to Buszkowski and Penn) underline the importance of the concept of the general form $\mathrm{GF}(D)$ (as discussed above), a CCG associated to a finite structure language D. Without giving details here, notice that it is further known that any reduced CCG G' consistent with D can be obtained from $\mathrm{GF}(D)$ by unification. Moreover, the size of $\mathrm{GF}(D)$ (and hence the size of G') is bounded by a polynomial over $\|D\|$. If D is a finite string language, then any finite structure language D' that yields D is of size polynomial in $\|D\|$. Hence, any $\mathrm{GF}(D')$ of interest is also of size polynomial in $\|D\|$.

Theorem 1. $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY *is* NP*-complete.*

Proof. NP-hardness of $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY is shown in [2] (Theorem 5.16, which holds for the case where $|\Sigma| = 3$). To see membership in NP, let (D, k) be an instance of $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY. The nondeterministic procedure we propose first generates some $G \in \mathcal{G}_{k-\text{valued}}$, by unifying types in $\mathrm{GF}(D)$ that are assigned to the same symbol. Then, for each structure $s \in D$, the procedure tests whether $s \in \mathrm{FL}(G)$ (which can be done in polynomial time). If (and only if) all these tests are passed, the algorithm returns YES. □

Theorem 2. $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY *is* NP*-complete.*

Proof. NP-hardness of $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY is shown in [2] (Theorem 5.32, which holds for the case where $k = 1$ and $|\Sigma|$ is unbounded). To see membership in NP, let (D, k) be an instance of $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY. The nondeterministic procedure we propose consists of two parts; first, for each string in D, a derivation is chosen. Then, the union of the resulting structures, D' (note that $\|D'\|$ is obviously polynomial in $\|D\|$), is used as a sample for learning from structures, so that some $G \in \mathcal{G}_{k-\text{valued}}$ is generated by unifying types in $\mathrm{GF}(D')$ that are assigned to the same symbol. Then, for each string $w \in D$, the procedure tests whether $w \in \mathrm{L}(G)$ (which can be done in polynomial time). If (and only if) all these tests are passed, the algorithm returns YES. □

We can actually sharpen the hardness assertion in the sense of parameterized complexity, by defining a polynomial-time transformation from HITTING SET to a dataset for a language in $\mathcal{FL}_{k-\text{valued}}$. Deciding that the data is consistent with a language in that class, i.e., $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY, then corresponds to deciding the existence of a cover of a specified size c. We will call any grammar that generates a language in the class consistent with the given input a *consistent grammar*.
We now define a construction that is based on these ideas.

Definition 1. *Let $hg(HG, c)$ be the algorithm that maps instances of the vertex cover problem for hypergraphs to samples of structure languages defined in the following way:*

The hypergraph $HG = (V, E)$ consists of a set V of vertices numbered $1, \ldots, v$ and a set E of (hyper)edges numbered $1, \ldots, e$. It is characterized by the functions $d(i)$, $1 \leq i \leq e$, which gives the degree of edge E_i, and $n(i, j)$, which gives the index of the jth vertex that edge i is incident on. The constant c specifies the maximal size of the cover.

The sample D output by hg is the smallest set that fulfills the following requirements:

For each i, $1 \leq i \leq e$, the structures
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_i, \underbrace{\mathtt{fa}(\mathtt{c}_i, \mathtt{x})) \ldots \mathtt{fa}(\mathtt{c}_i, \mathtt{x}))}_{d(i) \text{ times}}, \underbrace{\mathtt{u}_i), \ldots \mathtt{u}_i)}_{d(i)+1 \text{ times}}$ *and*
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_i, \mathtt{f}_{(i,1)}), \ldots \mathtt{f}_{(i,d(i))}), \mathtt{fa}(\mathtt{tt}, \mathtt{t})), \mathtt{s}_{(i,1)}), \ldots \mathtt{s}_{(i,d(i)-1)})$,
$\mathtt{fa}(\mathtt{ttt}, \mathtt{t}))$ *are in D. Additionally, for each j, $1 \leq j \leq d(i)$,*
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_i, G_{(j,1)}), \ldots G_{(j,d(i))}), T_{(j,1)}), \ldots T_{(j,d(i)+1)})$ *where $G_{(j,x)}$ is $\mathtt{fa}(\mathtt{v}_{n(i,x)}, \mathtt{x})$ if $x = j$ and $\mathtt{g}_{(i,x)}$ otherwise, and where $T_{(j,x)}$ is $\mathtt{fa}(\mathtt{t}_{j,j}, \mathtt{x})$ if $x = j$ or $x = j + 1$ and $\mathtt{t}_{(j,x)}$ otherwise.*

For each i, $1 \leq i \leq e$, $\mathtt{fa}(\mathtt{c}, \mathtt{fa}(\mathtt{c}_i, \mathtt{x}))$ is in D. If $c = 1$, the padding structure $\mathtt{ba}(\mathtt{x}, \mathtt{c})$ is in D.

For each i, $1 \leq i \leq c$, the padding structure $\mathtt{ba}(\mathtt{x}, \mathtt{c}_i)$ is in D.

For each i, $1 \leq i \leq v$, the padding structure $\mathtt{ba}(\mathtt{x}, \mathtt{v}_i)$ is in D.

For each $\mathtt{t}_{i,j}$, the padding structure $\mathtt{ba}(\mathtt{x}, \mathtt{t}_{i,j})$ is in D.

We add $k - 2$ padding structures for each $\mathtt{e}_i$, and $k - 1$ such structures for each $\mathtt{v}_i$, $\mathtt{c}_i$, and for $\mathtt{tt}$ and $\mathtt{ttt}$.

In order to make clear why the sample is built up in this way, we now discuss the types as they occur in any grammar in $\mathcal{G}_{k-\text{valued}}$ that is consistent with this sample.

Let $\Upsilon_i = C_{i,1}/\ldots C_{i,v}/(U_{s(1,i,0)}/U_{s(1,i,1)})/\ldots(U_{s(d(i),i,0)}/U_{s(d(i),i,1)})$,
$\Gamma_{n(i,j)} = G_{i,1}/\ldots G_{i,v}/\ (T_{t(1,i,0)}/T_{t(1,i,1)})/\ldots(T_{t(d(i),i,0)}/T_{t(d(i),i,1)})$,
$T_{t(k,i,0)} = T_{t(\ell,i,1)}$ if $\ell = n(i, j)$, and the $T_{t(\ell,i,j)}$s are distinct (primitive) types, otherwise.

Every type $G_{i,j}$, $i = j$, is equal to some type $\Delta_n(i, j)$. These are based strictly on alternating forward- and backward slashes, with the main operator always the backward slash. The type Δ_1 is $X\backslash t$. For any two u, v such that $u \neq v$, Δ_u and Δ_v are not unifiable. Note that this allows any two $\Gamma_{n(i,x)}$ and $\Gamma_{n(i,y)}$, $x \neq y$, to be unifiable, since the Δ-subtypes appear in different positions of these Γ terms.

Define $\Sigma_i = F_{i,1}/\ldots F_{i,v}/(S_{g(1,i,0)}/S_{g(1,i,1)})/\ldots(S_{g(v,i,0)}/S_{g(v,i,1)})$ for $1 \leq i \leq e$. From GF(D), the following grammar is derived by unifying all types assigned to $\mathtt{x}$. This simplifies the presentation without affecting the proof of W[2]-hardness in any way. Note that in the interest of clarity we omit type assignments to symbols $\mathtt{f}_{n(x,y)}$, $\mathtt{g}_{n(x,y)}$, $\mathtt{s}_{x,y}$, $\mathtt{t}_{x,y}$ and $\mathtt{u}_x$.

$$
\begin{aligned}
G' : \quad
&\mathtt{e}_1 \;\mapsto t/\Upsilon_1,\\
&\qquad t/\Gamma_{n(1,1)}, \ldots t/\Gamma_{n(1,d(1))},\\
&\qquad t/\Sigma_1,\\
&\qquad \mathit{Padding}\\
&\ldots\\
&\mathtt{e}_e \;\mapsto t/\Upsilon_e,\\
&\qquad t/\Gamma_{n(e,1)}, \ldots t/\Gamma_{n(e,d(e))},\\
&\qquad t/\Sigma_e,\\
&\qquad \mathit{Padding}\\
&\mathtt{v}_1 \;\mapsto \Delta_1, \mathit{Padding}\\
&\ldots\\
&\mathtt{v}_v \;\mapsto \Delta_v, \mathit{Padding}\\
&\mathtt{c}_1 \;\mapsto C_{1,1}/X, \ldots, C_{1,v}/X, C_1/X, \mathit{Padding}\\
&\ldots\\
&\mathtt{c}_e \;\mapsto C_{e,1}/X, \ldots, C_{e,v}/X, C_e/X, \mathit{Padding}\\
&\mathtt{c} \;\mapsto t/C_1, \ldots, t/C_e, \mathit{Padding}\\
&\mathtt{x} \;\mapsto X\\
&\mathtt{t} \;\mapsto t\\
&\mathtt{tt} \;\mapsto t/t, \mathit{Padding}\\
&\mathtt{ttt} \mapsto (t/t)/t, \mathit{Padding}
\end{aligned}
$$

Where, for $1 \leq i \leq e$, $j = n(i, d(i))$.
Note that *hg* runs in time polynomial in the size of the hypergraph. There are bounds on parameters of the grammar: given hypergraph $HG = (V, E)$ and stipulated size of the cover c, $k = \max(2, c)$, and $|\Sigma| = 5 + |V| + 2|E| + 2\sum_{i=1}^{|E|} d(i) + 2\sum_{i=1}^{|E|} d(i)^2$.
The construction works just for $k \geq 2$, but this does not affect the result in any way. Note that for $k = 1$, the consistency problem is known to be solvable in polynomial time.

Theorem 3. $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY *is* W[2]*-hard.*

Proof. By modifying the mentioned NP-hardness proofs, we show how to transform an instance (G, k) of HITTING SET to $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY, preserving the parameter. To be more precise, the HITTING SET problem can be reduced in polynomial time to finding a grammar consistent with structures D and in the class $\mathcal{G}_{k-\text{valued}}$. We achieve this using the algorithm *hg* as given in Definition 1.
Let hypergraph $HG = (V, E)$, $G = \text{GF}(hg(HG))$, and c such that a cover of size c exists for HG.
For any symbol $\mathtt{e}_i$, unification of all types $t/\Gamma_{n(i,1)}, \ldots t/\Gamma_{n(i,d(i))}$ to t/Σ_i will lead to a substitution such that $S_{g(1,i,0)} = S_{g(1,i,1)} = \ldots = S_{g(v,i,0)} = S_{g(v,i,1)}$. Since this is not possible, for each symbol $\mathtt{e}_i$, at most one of the types $t/\Gamma_{n(i,1)}, \ldots t/\Gamma_{n(i,d(i))}$ can be unified with t/Υ_i instead. For each i, only one of these t/Γ types can be chosen for this, since it will block

unification of t/Υ_i with any of the other t/Γ types: for any given i, the U types in Υ_i all have to be of the same type, and such a unification step will result in a substitution such that some Δ type will be substituted for all these U types.
For every i, the T types in Γ_i overlap: $T_{j,j}$ occurs twice in every Γ_i. Thus, they cannot all be unified with Σ_i, since the pair of the first and last S type in every Σ-type is not unifiable.
Hence, for each i, *exactly* one of the t/Γ types has to be unified with the t/Υ type, and the rest with the Σ type. This implies a substitution such that for each C_i, a Δ_ℓ is substituted such that $\ell = n(i,j)$, $1 \leq j \leq d(i)$. This corresponds to choosing vertex ℓ in the original hypergraph to cover edge i. Since, for all i, t/C_i is assigned to symbol c, and given the number of padding types assigned to this symbol (0 if $c \geq 2$, 1 if $c = 1$), the number of distinct Δ types that substitute for the C types can be no more than c. This proves that, if $k \geq c$, the answer to $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY is YES.
Let c and HG be such that a minimum cover for HG is of size $c' > c$, and let $G = \text{GF}(hg(HG))$ as before. Following the same line of reasoning as earlier in this proof, it is clear that for each C_i, a Δ_k must be substituted in order to obtain a consistent grammar that is in the class. Given the definition of hg, these Δ types correspond to one of the vertices that edge i is incident on. Given that $c' > c$, there are at least c' distinct such Δ types, and since for all i, t/C_i is assigned to c, at least c' distinct types are assigned to c in a consistent grammar, which thus cannot be in $\mathcal{G}_{k-\text{valued}}$ for $k = max(c, 2)$. Thus the answer to $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY is NO.
This proves that the answer to $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY for the sample $hg(HG, c)$ is the same as for HITTING SET for HG with c as size of the cover. Since the reduction hg runs in polynomial time, this proves W[2]-hardness. □

As for the problem $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY, note that $\mathcal{L}_{k-\text{valued}}$ contains Σ^* for $k \geq 2$, which immediately trivializes the problem. We refer to [6] for a proof of NP-hardness for the case $k = 1$. Notice that these results render the parameterization discussed in this section meaningless from the viewpoint of parameterized complexity.
As an aside, let us mention that in the literature (see Corollary 5.13 from [2], for example) also consistency questions related to the *least-k-valued* grammars and languages were considered. This means we are looking for the smallest k such that there is a grammar $G \in \mathcal{G}_{k-\text{valued}}$ and $D \subseteq \text{FL}(G)$. These problems are also known to be NP-hard, but it is an open question whether they belong to NP.

5 Reparameterizations

As already seen with the example of VERTEX COVER versus INDEPENDENT SET, basically the same problem can be parameterized in different ways, possibly leading to positive (FPT) results or to negative (W[2]-hardness) results. So it might be that other choices of parameterization may lead to FPT-algorithms.

One other natural choice of a parameter is the number of unification steps u needed to transform the general form of D into some k-valued grammar G such that $D \subseteq \mathrm{L}(G)$. This leads to problems like u-STEP $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY. The input to such a problem would be a triple (D, u, k), where D is the finite input sample. A variant could be UNIFORM u-STEP $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY, where the input would be (D, k), and the question would be whether there exists a u such that (D, u, k) is a YES-instance of u-STEP $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY. Hence, the inputs to UNIFORM u-STEP $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY and to $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY are the same.
Although the following is not stated explicitly in [1], it follows from Proposition 6.32 and the preceding description of algorithm VG_k, and the description of algorithm LVG (Section 6.3) in that book:

Theorem 4. *(D, k) is a YES-instance to $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY iff (D, k) is a YES-instance to* UNIFORM *u-STEP $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY.*

In conclusion, the uniform problem variants do not offer new insights. However, they immediately provide:

Corollary 2. UNIFORM *u-STEP $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY is* NP*-complete.* UNIFORM *u-STEP $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY is* W[2]*-hard.*

Instead of a single parameter, one could also consider two or more parameters. (Formally, this is captured by our definition by combining those multiple parameters into one single parameter.) So, the FPT-question of FIXED $|\Sigma|$ u-STEP $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY would be whether an algorithm exists that runs in time $f(u, k, |\Sigma|)p(\|D\|)$ for some function f and some polynomial p.

Theorem 5. FIXED $|\Sigma|$ *u-STEP $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY is in* FPT.

Proof. It is easy to see that, given that u bounds the number of unification steps, for a sample larger than $u + |\Sigma| \cdot k$, the answer to FIXED $|\Sigma|$ u-STEP $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY is always NO.
When the sample is smaller than this, we can obtain $\frac{1}{2} \cdot \|\mathrm{GF}(D)\| \cdot (\|\mathrm{GF}(D)\| - 1)$ as a bound for pairs of types and thus a bound on the size of the search-space of $\frac{u!}{(u+|\Sigma|\cdot k)!(|\Sigma|\cdot k)!}$. This is a constant, since u, k and $|\Sigma|$ are fixed. □

Theorem 6. FIXED $|\Sigma|$ *u-STEP $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY is in* FPT.

Proof. As in the case for structure languages, for a sample larger than $u + |\Sigma| \cdot k$ the answer to FIXED $|\Sigma|$ u-STEP $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY is NO. For smaller samples a consistent grammar may exist, so consider the number of derivations compatible with the strings in D. The length of the strings in D is upper bounded by $\|D\|$, and thus by $u + |\Sigma| \cdot k$, and the number of strings is $|D|$, which is upper bounded by $u + |\Sigma| \cdot k$, as well. Thus, given D, the number of possible structure samples D' is bounded by

$$\left(2^{u+|\Sigma|\cdot k-1} \frac{1}{u + |\Sigma| \cdot k} \binom{2(u + |\Sigma| \cdot k - 1)}{(u + |\Sigma| \cdot k - 1)}\right)^{u+|\Sigma|\cdot k}$$

which is a constant, since u, k and $|\Sigma|$ are fixed. For each of the possible D', FIXED $|\Sigma|$ u-STEP $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY can be considered, which is in FPT. □

One could also study the step number u (or other subsets of parameters) as another parameterization. One problem formulation could be the following one: Given a finite sample D and a categorial grammar G (and the parameter u), does there exist a sequence of at most u unification steps, starting from the general form GF(D) and leading to G? This might be an interesting subject for future study. However, note that the slightly more general problem of deciding the existence of such a sequence from some arbitrary given grammar (not necessarily in general form) is already at least as hard as the well-known GRAPH ISOMORPHISM problem for $u = 0$.[3] We can encode a graph into a grammar by assigning to one single symbol the types T_i/T_j for every edge from vertex i to vertex j in the graph (and assigning T_ℓ and t/T_ℓ for every vertex ℓ to avoid useless types). Let G_1 and G_2 be two such grammars encoding graphs $Graph_1$ and $Graph_2$, then $u = 0$ (i.e., an empty sequence of unification steps) just if there exists a renaming such that, when it is applied to G_1, G_2 is obtained. It is easy to see that this is only the case if $Graph_1$ and $Graph_2$ are isomorphic.

6 Consequences for the Complexity of Learning

We have studied the parameterized complexity, for several classes of categorial grammars, of selecting a grammar consistent with a given (string- or structure) sample. Our results have a direct consequence for the complexity of learning: if this problem is computable in polynomial time, then a learning algorithm with polynomial update time may exist.
Our complexity results are summarized in Table 6. As far as the authors are aware, these are the first such results for learning problems.

Problem	Complexity
$\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY	W[2]-hard
UNIFORM u-STEP $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY	W[2]-hard
UNIFORM u-STEP $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY	NP-complete
FIXED $\|\Sigma\|$ u-STEP $\mathcal{FL}_{k-\text{VALUED}}$-CONSISTENCY	FPT
FIXED $\|\Sigma\|$ u-STEP $\mathcal{L}_{k-\text{VALUED}}$-CONSISTENCY	FPT

From a technical point of view, it would be nice to complement our W[2]-hardness result (Theorem 3) by demonstrating membership in W[2]. A natural idea would be to design a multi-tape Turing machine with one

[3] Though it is not known if GRAPH ISOMORPHISM is NP-complete, this problem is also believed not to be solvable in polynomial time, see [8,9].

tape storing or counting the rule (applications) for each symbol. However, it is not clear if such a Turing machine would need only $f(k)$ many steps to decide consistency. Such a question might also be interpreted in the direction of parallelizability of derivations in categorial grammars. We are not aware of any such study for this type of mechanisms, but we would like to point to the fact that studies in this direction were undertaken for the weakly equivalent mechanism of context-free grammars, see [10] and the references therein.

The obvious interpretation of our positive (FPT) results would be that, as long as the parameters k, u, and $|\Sigma|$ are kept low, the classes are efficiently learnable. The last parameter is the most problematic, since for typical (NLP) applications the lexicon is very large. Thus, our analysis suggests the approach of choosing the total number of distinct types in the grammar as a parameter, and keeping this number low.

It would be interesting to study the consistency problem for other language class hierarchies, where each class has finite elasticity. One such example might be those based on elementary formal systems as examined by Moriyama and Sato [11]. This would be an interesting topic for future research.

References

1. Kanazawa, M.: Learnable Classes of Categorial Grammars. PhD, CSLI (1998)
2. Costa Florêncio, C.: Learning Categorial Grammars. PhD thesis, Universiteit Utrecht, The Netherlands (2003)
3. Buszkowski, W.: Discovery procedures for categorial grammars. In Klein, E., van Benthem, J., eds.: Categories, Polymorphism and Unification. University of Amsterdam (1987)
4. Buszkowski, W., Penn, G.: Categorial grammars determined from linguistic data by unification. Studia Logica **49** (1990) 431–454
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
6. Costa Florêncio, C.: Consistent identification in the limit of rigid grammars from strings is NP-hard. In Adriaans, P., Fernau, H., van Zaanen, M., eds.: Grammatical Inference: Algorithms and Applications; 6th International Colloquium, ICGI. Volume 2484 of LNCS/LNAI., Springer (2002) 49–62
7. Gallai, T.: Über extreme Punkt- und Kantenmengen. Ann. Univ. Sci. Budapest, Eötvös Sect. Math. **2** (1959) 133–138
8. Johnson, D.S.: The NP-completeness column. ACM Transactions on Algorithms **1**(1) (2005) 160–176
9. Köbler, J., Schöning, U., Torán, J.: Graph Isomorphism Problem: The Structural Complexity. Birkhäuser Verlag (1993)
10. Reinhardt, K.: A parallel context-free derivation hierarchy. In Ciobanu, G., Păun, G., eds.: FCT. Volume 1684 of LNCS., Springer (1999) 441–450
11. Moriyama, T., Sato, M.: Properties of language classes with finite elasticity. In Jantke, K.P., et al., eds.: 4th Workshop on Algorithmic Learning Theory ALT'93. Volume 744 of LNCS/LNAI. (1993) 187–196